Practica 1

biblioteca gui

## Java 2Se y Aplicaciones Web con Servlets y Jsp 2016

# 1 gestión, requisitos y objetivos

## Análisis

El punto de partida para la practica es el codigo de la biblioteca en su revision 131, presente en:

<http://62.204.199.127/JAVA_UNED/bea/browser/biblioteca_Mockito>

El objetivo es completar en la medida de lo posible las funcionalidades opcionales marcadas en la práctica general.

### Especificación de requisitos

Enumeramos a continuación los requisitos a completar en la práctica, y los identificaremos adecuadamente para poder referenciarlos en el resto del documento. Para ello, incluiremos el identificador del requisito siguiendo el esquema [XXX-N] donde XXX implica el ámbito del requisito, y N un número entero.

#### Requisitos globales

* [GBL-1] Soporte para conexion a Oracle: Se requiere que la aplicación pueda correr sobre el SGBD Relacional Oracle, de manera q sea transparente para el usuario el uso de mysql o de Oracle en la ejecución de la aplicación.
* [GBL-2] Creación de menú CRUD en menú popup contextual para todos los objetos de negocio: Se requiere que, al seleccionar una fila y pulsar sobre el botón derecho, se muestre al usuario un menú en el que se le permita lanzar las operaciones CRUD sobre la fila seleccionada. El objetivo es lanzar programáticamente el evento provocado por el usuario al usar el menú CRUD q ahora se muestra en la parte superior de cada interfaz asociadoa la operativa de cada objeto de negocio (socios,prestamos y libros)
* [GBL-3]Internacionalización de la aplicación: Se requiere la adición de un nuevo idioma a la aplicación, en este caso el idioma Francés
* [GBL-4]Cierre ordenado de la aplicación: Se requiere que si el usuario decide cerrar la aplicación, se muestre un diálogo modal preguntándole si está seguro, y en caso afirmativo, realizar un cierre controlado de los recursos adquiridos por la aplicación (en este caso, la conexión a la base de datos).
* [GBL-5]Pruebas unitarias: Se requiere el uso de JUnit para la ejecución de un juego de pruebas unitarias, y opcional el uso de Mockito.
* [GBL-6] Creación de un HOW TO y documentación para el uso de JUnit y Mockito, de manera q pueda utilizarse por los alumnos de otros años de este mismo master, y q debe quedar incluido en la wiki del espacio 'alef' del máster.
* [GBL-7] Inclusión de una presentación en slideshare a modo de resumen del presente texto.

#### Requisitos asociados a socios

* [SCS-1]Inclusión de una ficha para la modificación de SOCIOS: Actualmente está en un dialogo modal, se requiere al sistema q la modificación de socios se haga en una ficha integrada en el interfaz de socios.

#### Requisitos asociados a prestamos

* [PRT-1]Inclusión de una ficha para la inserción de PRESTAMOS: Actualmente está en un dialogo modal, se requiere al sistema q la inserción de prestamos se haga en una ficha integrada en el interfaz de prestamos.
* [PRT-2]Inclusión de una ficha para la modificación de PRESTAMOS: Actualmente está en un dialogo modal, se requiere al sistema q la modificación de prestamos se haga en una ficha integrada en el interfaz de prestamos.
* [PRT-3]En los dos casos anteriores, se requiere al sistema q tanto libros como socios aparezcan en un combo-box y ordenados segun lo siguiente:
  + Campo socio desplegable ordenado alfabéticamente por "apellidos, nombre".
  + Campo libro desplegable ordenado alfabéticamente por "título".

#### requisitos asociados a libros

* [LBR-1]Inclusión de una ficha para la inserción de PRESTAMOS: Actualmente está en un dialogo modal, se requiere al sistema q la inserción de prestamos se haga en una ficha integrada en el interfaz de prestamos.
* [LBR-2]Inclusión de una ficha para la modificación de PRESTAMOS: Actualmente está en un dialogo modal, se requiere al sistema q la modificación de prestamos se haga en una ficha integrada en el interfaz de prestamos.

## metodología de trabajo

En la medida de nuestras posibilidades, se ha intentado llevar a cabo una metodología SCRUM, con un coordinador o Scrum Master, el equipo de trabajo y un responsable de producto (en este caso el profesor).

### roles

* Equipo: Compuesto por:
  + Carlos Almarcha - en adelante 'calmarcha'
  + Carlos Larrondo - en adelante 'clarrondo'
  + Federico Graziano - en adelante 'fgraziano'
  + Ismael Martín - en adelante 'imartin'
  + Ignacio Cerezo - en adelante 'icerezo'
* Responsable de producto
  + Carlos Luis Sánchez
* Scrum Master
  + José Barba - en adelante 'jbarba'

### herramientas de coordinacion utilizadas

* Codificación: Para el registro del código asociado a cada requisito hemos utilizado SVN, aunque hemos tenido problemas con el repositorio central q nos ha obligado a utilizar una estrategia diferente, que explicaremos en una sección posterior de este documento.
* Coordinación: Para la representación y asignación de tareas se ha utilizado RedMine, sobre un hosting gratuito en www.hostedredmine.org. El equipo está definido en esa instancia de RedMine, así como el proyecto y cada una de las tareas y sus asignaciones a miembros del equipo. Tiene muchas carencias, pero para un reparto básico de tareas y una mínima estructura es suficiente.

### Detalle de la metodologia usada

#### uso de redmine

La URL a la que accedemos para entrar en redmine es:

<http://www.hostedredmine.com>

* Proyectos
  + Se creará un proyecto en RedMine para representar cada práctica.
  + En cada proyecto habrá un Manager (Scrum Master) y el resto serán Developers (Equipo)
* Tareas
  + Las tareas serán determinadas por el Scrum Master y asignadas a cada miembro del equipo.
  + Las tareas se representarán como 'Issues' de tipo 'Task', y serán asginadas a acada miembro del equipo para su registro y control
  + El ScrumMaster podrá cambiar la asignación de tareas bajo demanda del equipo.
  + La definición de qué hay que hacer en cada tarea es del Scrum Master, dicho enunciado deberá ser claro, sencillo y explicar claramente qué se desea
  + Bajo criterio del responsable de la tarea, éste podrá crear subtareas (add subtask en la edición de tareas) para reflejar avances o cualquier aspecto que considere oportuno.
  + Cada tarea tiene la opción de incluir notas, que puede utilizarse para registrar información relevante, avance de cada tarea y dudas sobre la tarea. Las preguntas, dudas y problemas q se quieran tratar con el ScrumMaster se tratarán incluyendo una nota en la tarea.
  + El ScrumMaster es responsable de monitorizar la actividad de cada tarea asignada, detectando las necesidades y dudas de cada miembro y resolviendolas. Para ello, será seguidor de todas las tareas q asigne a miembros del equipo. Al ser seguidor, RedMine enviará un mail con el contenido de la actualización de la tarea, dándole una oportunidad de realizar el seguimiento.
* Distribucion de tiempo semanal del equipo
  + Lunes -> Reunión semanal con el docente UNED, a petición y organización del docente UNED
  + Martes-Miercoles -> Asingación de tareas por parte del ScrumMaster al equipo
  + Miercoles -> 'Reunión Periódica' con el equipo, a petición de ScrumMaster (SPRINT)
  + Jueves-Domingo -> Tiempo efectivo de desarrollo de tareas asignadas por el ScrumMaster al equipo.
* Reuniones y Contactos del equipo
  + Se establece una 'reunion periódica', situada los Miercoles.
    - El objetivo de la reunión es discutir y aclarar las dudas que cada miembro del equipo pueda tener con las tareas asignadas para esa semana de trabajo, de manera que dichas dudas surjan antes del tiempo efectivo de desarrollo de tareas (definido más abajo)
    - El solicitante de la reunión es el ScrumMaster, que será el encargado de poner los medios (propuesto HangOuts) para el contacto del equipo.
    - Antes de la reunión el ScrumMaster debe haber asignado todas las tareas a cada miembro del equipo.
    - Tras la reunión, cada miembro del equipo tiene claras sus tareas y responsabilidades durante el tiempo efectivo de desarrollo de tareas (definido más abajo)
    - Duracion estimada: 15 a 30 minutos.
    - Asistencia: Voluntaria, no se requiere la asistencia a aquellos miembros que no tengan dudas sobre el trabajo asignado.
  + Se establece un contacto 'bajo demanda', situada en cualquier momento de la semana de trabajo:
    - Este contacto se utilizará cuando se quiera llamar la atención al scrum master por parte de un miembro del equipo sobre un particular concreto.
    - Si el ScrumMaster lo considera necesario, podría registrar y hacer un seguimiento del contacto realizado.

#### Gestión y uso de SVN

Debido a un problema de gestión interno en los servidores de la UNED, el uso del SVN ha sido restringido al propio ScrumMaster como gestor y coordinador del código desarrollado por cada uno de los miembros del equipo. Esto no invalida la estrategia utilizada, sólo la hace más tediosa por parte del Scrum Master al tener q usar otro medio distnto al propio SVN para comunicarse con cada miembro del equipo.

##### Estructura física del repositorio

La estructura utilizada en el repositorio SVN (en este caso, local al ScrumMaster y sin acceso al mismo por parte del equipo) fue la siguiente:

<root>/practica1-BibliotecaGUI

|- branches

|- agomez

|- calmarcha

|- clarrondo

|- fgraziano

|- icerezo

|- imartin

|- jbarba

|- dist

|- trunk

Siendo el propósito de cada directorio:

* branches: Aquí incluimos las ramas de desarrollo, incluyendo un directorio por cada integrante del equipo. Asumimos q cada desarrollo de cada integrante será una copia del trunk en una revision concreta,en la carpeta del ejecutor de la tarea asignada.
* dist: Directorio reservado para las entregas asociadas a esta práctica. Aquí se guarda exactamente lo q se entrega a cliente.
* trunk: Directorio maestro con el código fuente actual de la práctica. En un estado inicial, aquí se incluye exactamente el código de partida (si lo hay, como este caso). En un estado final, este directorio debe contener TODAS las modificaciones realizadas por cada integrante del equipo, siendo la agregación de todo el trabajo de éstos.

##### Herramientas utilizadas

Para esta gestión se utilizó:

* Repositorio SVN sobre ArchLinux versión 1.9.0-1
* Cliente para windows TortoiseSVN versión 1.9

##### Ciclo de vida de una tarea de codificación en el repositorio SVN

Cada vez q se asigna una tarea que requiere de desarrollo de código, debe hacerse una copia del repo (crear un branch, usando la operación CopyTo) y crear una rama en el directorio <root>/branches/<usuairo>/<id\_tarea>, donde 'id\_tarea' puede ser el identificador de RedMine de la tarea asignada más un texto aclarativo para referencia (en este caso no lo hemos usado, y se echa de menos al revisar el repo).

Es necesario recalcar que cada tarea SIEMPRE parte de un código fuente en una revisión concreta del trunk, código sobre el q deben implementarse, probarse y validarse los cambios incluidos en el ámbito de la tarea.

Una vez validados los cambios en el ámbito de la tarea asignada en la rama creada (<root>/branches/<id\_usuario>/<id\_tarea>), estos cambios se deben integrar en el trunk de forma coordinada. Esto es, si el trunk ha evolucionado desde q se creó la rama q implementa la tarea actual, es necesario incluir en el trunk sólo los cambios de esta rama, para evitar perder cambios en el trunk ya validados.

Para hacer esto se utiliza la operación 'merge', poniendo especial cuidado en qué 'fusionar' en el trunk para no perder cambios importantes.

Esta forma de trabajo permite desacoplar a los distintos miembros del equipo para que entre ellos NO se creen dependencias, pero complica el papel del ScrumMaster como encargado de coordinar que no haya problemas de integración entre ellos, y que 'no se pisen unos a otros' los cambios realizados.

##### Adecuación al problema con el SVN de la UNED

Para adecuarnos a no tener un SVN compartido, la estrategia ha sido esta:

* El ScrumMaster crea en su repositorio SVN local la estructura de trabajo, cargando en el trunk el estado inicial del código (referenciado al principio de este doc)
* El SM crea una rama por cada tarea a realizar, emplazándola en el directorio de trabajo correspondiente de cada responsable.
* El SM envía por mail una copia de dicha rama a cada miembro, para que trabaje en la solución a su tarea
* Cada vez q un miembro del equipo indica q ha finalizado una tarea:
  + El responsable de la tarea envía el proyecto SIN librerías y SIN clases (fuentes y configuracion) al SM.
  + El SM vuelca esa copia contra un checkout de la rama correspondiente en local, pero NO acepta los cambios
  + el SM comprueba q los requisitos de la tarea se cumplen. En caso contrario, comunica el error al responsable y la gestión termina.
  + Si el cambio es correcto, el SM realiza un commit sobre el checkout y registra en los comentarios al commit el medio y la fecha exacta en la q recibió esa versión del código (puede usarse otro medio)

Esta estrategia permite desacoplar a cada miembro, pero llegado un momento es necesario que los cambios de una rama se pasen al trunk global:

* El SM decide pasar una rama al trunk global
* Para ello, realiza un checkout del trunk y realiza un merge sobre dicho checkout de la rama deseada.
* Antes de realizar un commit sobre el trunk, repasa cuidadosamente los cambios que dicho merge incluirían en el trunk, para evitar perder cambios.

##### Cambios globales y locales

Un problema q nos hemos encontrado y q ha complicado mucho la gestión es el hecho de que ciertos miembros del equipo trabajen sobre áreas globales de la aplicación,y otros en áreas muy focalizadas. Concretamente, el caso del paso a Oracle de la aplicación (cambio global) ha impactado en todos los miembros del equipo (todos usan en algun momento la BBDD).

Para no tener q esperar a que todos terminasen sus tareas, se decidio lo siguiente:

* El SM pasó al trunk los cambios asociados al acceso a Oracle al trunk.
  + en este punto el trunk ya estaba más avanzado q cada código fuente enviado a cada responsable.
* El SM volvió a crear una rama por cada tarea, asociada a cada usuario (pero ahora con el código de soporte a Oracle)
* El SM solicitó a cada miembro que le pasase su código.
* Por cada código recibido
  + El SM hace un checkout de la nueva rama (la q ya tiene el código Oracle)
  + El SM machaca los fuentes de la rama con los recibidos de cada miembro.
  + El SM vía uso de Tortoise incluye en la rama los cambios de cada miembro del grupo.
  + El SM comita los cambios contra el SVN, una vez probados.

En este punto, el SM ya tiene por cada rama los cambios globales y los realizados por cada miembro. Sólo queda enviar los fuentes (sólo los fuentes) a cada miembro para q siga trabajando.

Notar q este problema lo hubiéramos tenido igual en el caso de usar un SVN centralizado, ya q es inherente al hecho de permitir una paralelización en el trabajo del equipo.

##### tareas de replicacion

Para aprovechar al máximo los recursos se han identificado tareas repetitivas, de forma que se ha resuelto la tarea una vez, y la solución se ha pasado al equipo para su implementación. Esto se ha hecho con el requisito [GBL-2] de creación de un popup contextual para operaciones CRUD.

La idea fué implementarlo para socios (tarea implementada por agomez), realizar un exámen de los cambios a nivel de código (usando SVN), y pasar al resto del equipo qué tenían que incluir, facilitándoles qué clases y en qué punto había cambiado cada clase para implementar el menú. Para ello se utilizó WinMerge y dos revisiones del código, una antes (version origen) y otra después (version deseada) con la implementación.

## Reparto de tareas

Durante la evolución de la práctica este ha sido el reparto de tareas

* Requisitos Globales
  + [GLB-1] Soporte Oracle
    - asignado a: agomez
    - comentarios: Al ser un cambio global a la aplicación sus cambios se han tenido q coordinar con los del resto por medio del scrum master
  + [GBL-2] Menu contextual CRUD: Este requisito se ha dividido en varias partes:
    - Primero se hace para Socios: Realizado por agomez
    - Y se traspasan los cambios al resto
      * doc de requisito con los cambios: jbarba
      * implementación para Libros: icerezo
      * implementación para préstamos: fgraciano e imartin.
  + [GBL-3] Internacionalización:
    - asignado a clarrondo
  + [GBL-4] Cierre ordenado
    - asignado a calmarcha
  + [GBL-5] Uso de JUnit-Mockito
    - este requisito global está incluido en cada tarea.
  + [GBL-6] Howto para JUnit-Mockito para subir a wiki
    - asignado a :jbarba
  + [GBL-7] Presentacion slideshare
    - cada miembro del equipo crea su parte
    - jbarba las agrupa y sube a slideshare
* Requisitos asociados a SOCIOS
  + [SCS-1] Ficha para modificacion de socios
    - asignado a: clarrondo
* Requisitos asociados a Prestamos
  + [PRT-1] Ficha para insercion de prestamos
    - asingada a: imartin
  + [PRT-2] Ficha para modificacion de prestamos
    - asignada a: fgraziano
  + [PRT-3] Combos ordenados para socios y libros:
    - asignada a: fgraciano e imartin, cada uno en su interfaz
* Requisitos asociados a Libros
  + [LBR-1] Ficha para insercion de libros
    - asignada a :icerezo
  + [LBR-2] Ficha para modificacion de libros
    - asignada a :icerezo

# detalle técnico

En esta sección se detalla cada requisito implementado, atendiendo a la estrategia de solución adoptada y sus detalles de implementación.

## [GBL-1] Soporte Oracle

##### **Especificación de requisito**

Se debe incluir una nueva funcionalidad que permita conectar la aplicación a una base de datos Oracle además de la posibilidad actual de realizarlo contra MySQL. Para ello además de implementar el driver de Oracle, se deberán revisar y modificar si procede las distintas clases que tienen acceso a datos.

##### **Estrategia de solución usada**

La solución entregada se estructura en dos partes bien diferenciadas. Por un lado, se ha modificado la clase que realiza la conexión a base de datos, para que permita conectarse contra una base de datos Oracle, esto se harámediante un fichero de configuración.

Por otra parte, se han tenido que modificar las clases que implementan DAO tanto en el módulo de socios como en el de libros. La solución adoptada ha sido extender estas clases ya que se ha tenido que modificar el método de inserción debido a que la generación de identificadores es diferente en MySQL que en Oracle. Por lo tanto, se ha sobrescrito este método, adaptándolo a la nueva funcionalidad.

Para la generación de identificadores en Oracle se ha creado una nueva clase, la cual contiene un método estático que devuelve el valor de la secuencia que corresponde.

##### **Modificaciones realizadas sobre código existente**

Clase ConnectionManager (modificación)

Esta clase es la encargada de gestionar las conexiones contra base de datos. Ha sido modificada para que realice la conexión dependiendo de la propiedad dbms que se encuentra dentro del fichero Config.properties. En caso de que este valor sea mysql, realizara la conexión del modo habitual. Si su valor fuese oracle, generara la cadena de conexión para Oracle, permitiendo realizarse tanto mediante servicename como mediante SID.

Clase CustomersManagementController(modificación)

Se ha modificado el constructor de la clase para que instancie el objeto ICustomersDAO a partir de la clase CustomersDAOImpl o CustomersDAOImplOracle, dependiendo del valor de la propiedad dbms del fichero de configuración.

Clase BooksManagementController(modificación)

Se ha modificado el constructor de la clase para que instancie el objeto IBooksDAO a partir de la clase BooksDAOImpl o BooksDAOImplOracle, dependiendo del valor de la propiedad dbmsdel fichero de configuración.

CustomersDAOImplOracle(nueva clase)

Esta nueva clase extiende de CustomersDAOImpl y sobrescribe el método insert ya que como se ha comentado antes es diferente la inserción de datos, ya que en MySQL el identificador se autogenera, mientras que en Oracle este se debe obtener a partir del valor de un objeto secuencia. Por lo tanto, en este nuevo método, asignamos el IdSocio mediante el valor obtenido en UtilsOracle.generaSecuenciapara SEQ\_SOCIO.

BooksDAOImplOracle(nueva clase)

Esta nueva clase extiende de BooksDAOImpl y sobrescribe el método insert ya que como se ha comentado antes es diferente la inserción de datos, ya que en MySQL el identificador se autogenera, mientras que en Oracle este se debe obtener a partir del valor de un objeto secuencia. Por lo tanto, en este nuevo método, asignamos el IdSocio mediante el valor obtenido en UtilsOracle.generaSecuencia para SEQ\_LIBRO.

UtilsOracle(nueva clase)

La creación de esta clase surge a partir de la necesidad de obtener los identificadores a través de secuencias (objeto Oracle). Por ello se crea un nuevo método estático al cual se le envía la conexión actual, así como el nombre de la secuencia sobre la cual se desea obtener el nuevo valor. Esta conectará con la base de datos y consultará el valor NEXTVAL de la misma, el cual será devuelto por la función.

##### **Dificultades encontradas**

Para la realización de este cambio se tuvo que montar un servidor Oracle en el cual generar un esquema similar al proporcionado para MySQL.

El mayor problema surgió como se ha comentado anteriormente con la generación de los identificadores para las tablas de socios y libros. Se decidió resolver mediante la creación de Secuencias ya que es la herramienta que se utiliza habitualmente cuando se trabaja en bases de datos Oracle.

##### **Pruebas Unitarias**

Para probar la implementación de este cambio se realizaron las siguientes pruebas mediante la utilización de JUnit y Mockito:

* Se realiza Mock sobre la clase UtilsOracle y se comprueba que el valor retornado por el método estático generaSecuencia es el esperado.
* Se comprueba que se accede una vez a través del método generaSecuencia.

Cabe destacar que ha sido necesaria la utilización de la librería PowerMockito, ya que la versión estándar no permite la realización de pruebas con métodos estáticos. Estas pruebas se han realizado tanto para BooksDAOImplOracle como para CustomersDAOImplOracle.

## [GBL-2] Menu contextual CRUD

##### **Especificación de requisito**

Se debe incluir una nueva funcionalidad que permita realizar las operaciones CRUD en el componente de socios mediante un menú contextual que se activara pulsando el botón derecho del ratón. Las operaciones que podrán realizarse son las mismas que las existentes en la botonera superior (añadir, editar y eliminar).

##### **Estrategia de solución usada**

Para la implementación de este requisito se ha optado por reutilizar la lógica existente en la botonera superior, de esta manera únicamente se ha desarrollado la parte relacionada con la vista, creando para ello los componentes necesarios y realizando llamadas a los métodos ya existentes

##### **Modificaciones realizadas sobre código existente**

Clase CustomerManagementView

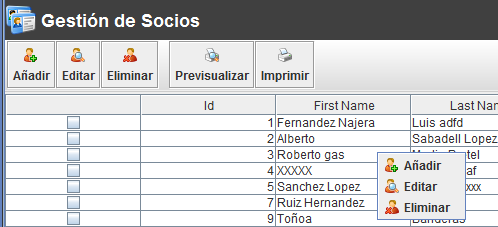
Se ha modificado la clase relaciona con la vista del componente de socios incluyendo un nuevo elemento de tipo JPopMenu de la librería Swing el cual a su vez está formado por tres elementos de tipo JMenuItem los cuales representan las tres posibles acciones a realizar. Estos tres elementos tienen a su vez un listener que controla el clicado sobre cada uno de ellos.

Además de esto se configura la parte estética del menú, incluyendo iconos a los distintos ítems, así como toda la lógica relacionada con la internacionalización.

Clase CustomerManagementController

En esta clase se ha modificado el método que gestiona los diferentes listeners de la vista clonando la lógica existente para la botonera superior, es decir, se llamaran a las mismas acciones que con los botones. Por ejemplo, el botón de Añadir Socio, ejecuta la acción ACTION\_ADD al igual que el ítem del menú contextual

El aspecto final del menú se muestra en la siguiente figura:



## Prestamos, requisitos [PRT-1] [PRT-2] [PRT-3]

##### **ESPECIFICACIÓN DE REQUISITO**

Eliminación ventana externa por ficha interna

Las tres funcionalidades principales de la Biblioteca de Préstamos están relacionadas con las tablas de su base de datos, por tanto con la información de Socios, Libros y Préstamos.

Para poder actualizar la información de dichas tablas, el software original utilizaba una ventana adicional objeto de la clase XXXXXXXX. El requisito determina la eliminación del uso de esta ventana modal. Para ello y ya centrándonos en la funcionalidad de Préstamos, utilizando la actual ventana de Préstamos objeto de la clase LoansManagemenView.

El requisito se cumplirá con la incorporación de 8 elementos visuales swing dentro de la ventana de Préstamos objeto de la clase LoansManagemenView que permitirán al usuario gestor de la Biblioteca entregar la información necesaria para Añadir o Editar un préstamo.

Listado de nuevos elementos:

* 4 JLabel para indentificar al resto de elementos.
* 2 JComboBox para recoger las listas cerradas de Socios y Libros que pueden formar parte de un Préstamo.
* 2 JDateChooser para recoger las fechas de Inicio y Fin de un préstamo.

No se necesita atender los eventos de estos componentes puesto que el usuario indicará la gestión de la información desde los botones componentes de la botonera.

Control de la funcionalidad desde la botonera

La conformación de la botonera para la ventana de Préstamos está formada originalmente por los JButton siguientes: Añadir, Editar, Eliminar, Preview y Print.

Solo "Eliminar" funciona plenamente seleccionando sobre la tabla de Préstamos aquellos a ser borrado de base de datos. El resto de botones están velados puesto que no tienen funcionalidad.

Del resto de botones daremos funcionalidad a "Añadir" y "Editar". Para ello generaremos sus eventos en la clase "LoansManagementView" y atendiendolos desde la clase "LoansManagementController".

Necesitamos dos nuevos botones para un correcto manejo de la funcionalidad: "Aceptar" y "Cancelar".

Para conocer si al Aceptar se debe añadir o corregir la edición de un préstamo, utilizaremos una variable de control "ultimaaccion". Esto se hará en la clase de Control de Préstamos LoansManagementController.

Control de errores de escritura en los componentes de la ficha

Como requisitos tenemos que la aplicación no debe aceptar que el usuario introduzca valores en blanco. Los errores que pueden producirse a priori son:

* El usuario introduce fechas en blanco.
* El usuario edita sin seleccionar ninguna columna.

Para cumplir con este requisito generaremos métodos en la clase vista "LoansManagementView" que comprueben que el valor de los componentes es correcto y que hay alguna fila seleccionada de la tabla.

Estos métodos serán llamados desde la clase "LoansManagementController" para utilizarlos en la lógica de control de la actualización de Préstamos.

Los métodos son:

* public boolean isFechaValida() //Detecta fechas en blanco.
* public boolean isSelectedRow() //Detecta préstamo no seleccionado.

##### **Estrategia de solución usada**

Adaptación sobre las clases ya realizadas

El primer paso en cualquier estrategia que parta de unas clases de partida es su estudio. De ese estudio decidiremos que parte del desarrollo inicial nos es válido y que parte necesitamos redefinir o cambiar.

Para nuestro caso la mayor parte del desarrollo nos es válida. La separación impuesta por el patrón MVC está claramente definida. En la parte de Préstamos no estaba desarrollada la ventana externa a eliminar pero si que necesitábamos incorporar código para cubrir funcionalidad.

Las clases que eran suceptibles de modificar eran las clases del paquete es.csc.biblioteca.loans.gui: LoansManagementView y LoansManagementController.

Y las clases para manejar los modelos CustomersDAOImpl y BooksDAOImpl.

Decidimos modificar exclusivamente las clases del paquete es.csc.biblioteca.loans.gui para no interferir en el trabajo de los demás. La consecuencia será que tendremos un par de métodos en la clase controladora "LoansManagementController" que bien podrían estar en las clases de los paquetes de modelo.

Acceso al modelo completo

En nuestro caso, la funcionalidad de Préstamos, a diferencia de la funcionalidad de Socios o de Libros, necesita el acceso completo al modelo. No solo se necesita acceder al modelo de Préstamos, sino también al modelo de Socios y Libros puesto que se debe mostrar al usuario información sobre estos modelos a través de los componentes JComboBox.

Para acceder al modelo de préstamos inicialmente se importaron las clases:

* import es.csc.biblioteca.loans.dao.ILoansDAO;
* import es.csc.biblioteca.loans.dao.LoansDAOException;
* import es.csc.biblioteca.loans.dao.LoansDAOImpl;

Para acceder al resto de modelos se incluyó las siguientes importaciones:

* import es.csc.biblioteca.customers.dao.\*;
* import es.csc.biblioteca.books.dao.\*;
* import es.csc.biblioteca.loans.dao.LoanDTO;

Mientras la ventana de Préstamos esté abierta se mantendrá la conexión con los modelos. Esto se realizó dentro de la clase controladora ampliando atributos y modificando su método constructor.

private final ILoansDAO dao; //Acceso inicial al modelo Préstamos.

private final ICustomersDAO custdao; //Ampliación para acceso al modelo Socios.

private final IBooksDAO bookdao; //Ampliación para acceso al modelo Libros.

public LoansManagementController(Connection connection) throws LoansDAOException {

if(ConnectionsManager.dbms.equals("oracle")){

this.dao = new LoansDAOImplOracle(connection);

}

else{

this.dao = new LoansDAOImpl(connection);

}

this.model = new LoansTableModel(this.dao, this.dao.findAll());

this.view = new LoansManagementView();

initView();

addListeners();

this.view.setVisible(true);

LocaleManager.getInstance().addObserver(this);

this.bookdao = new BooksDAOImpl (connection); //Conexión al modelo Book.

this.custdao = new CustomersDAOImpl (connection); //Conexión al modelo Customer.

}

Incorporación de nuevos elementos visuales

El siguiente paso era identificar los componentes visuales necesarios para la ficha e intentar ver si el uso de los botones ya generados sería suficiente.

Vimos que sería necesario incluir etiquetas que identificaban los campos modificables y que para gestionar los préstamos necesitábamos 2 JComboBox y 2 JDateChooser.

En cuanto a los botones, faltaban los botones para Aceptar y Cancelar.

Generación de métodos y eventos

Una vez introducidos los componentes visuales, el siguiente paso era generar los eventos desde la clase vista LoansManagementView para los botones. Estos eventos deberán ser atendidos desde la clase controladora LoansManagementController.

Dentro de la clase controladora también fuimos generando métodos para atender la pulsación de botones. Al principio eran métodos vacíos con algún código de salida por consola para conocer la evolución del desarrollo. Finalmente los métodos de control realizan llamadas a métodos ubicados en las clases vista y modelo.

Generación de eventos en la clase vista LoansManagementView:

*public void setAceptListener(ActionListener l, String command) {*

*this.btnAcept.addActionListener(l);*

*this.btnAcept.setActionCommand(command);*

*}*

*public void setCancelListener(ActionListener l, String command) {*

*this.btnCancel.addActionListener(l);*

*this.btnCancel.setActionCommand(command);*

*}*

El código anterior corresponde a los métodos añadidos para generar los eventos de los botones Aceptar y Cancelar. Las llamadas a estos métodos suman un escuchador del evento e indican el texto indentificador del evento.

El escuchador será la clase controladora LoansManagementController configurado mediante el método siguiente:

*private static final String ACTION\_ADD = "Add";*

*private static final String ACTION\_EDIT = "Edit";*

*private static final String ACTION\_DELETE = "Delete";*

*private static final String ACTION\_PRINT = "Print";*

*private static final String ACTION\_PREVIEW = "Preview";*

*private static final String ACTION\_ACEPT = "Acept";*

*private static final String ACTION\_CANCEL = "Cancel";*

*private void addListeners() {*

*this.view.setAddListener(this, ACTION\_ADD);*

*this.view.setEditListener(this, ACTION\_EDIT);*

*this.view.setDeleteListener(this, ACTION\_DELETE);*

*this.view.setPrintListener(this, ACTION\_PRINT);*

*this.view.setPreviewListener(this, ACTION\_PREVIEW);*

*this.view.setAceptListener(this, ACTION\_ACEPT);*

*this.view.setCancelListener(this, ACTION\_CANCEL);*

*}*

Y el método que atiende los eventos en la clase controladora:

*@Override*

*public void actionPerformed(ActionEvent e) {*

*switch (e.getActionCommand()) {*

*case ACTION\_ADD:*

*UltimaAccion = AccionLoans.ADD;*

*this.view.setBotones(1);*

*rellenaCombos();*

*break;*

*case ACTION\_EDIT:*

*if (this.view.isSelectedRow()){*

*UltimaAccion = AccionLoans.EDIT;*

*this.view.setBotones(1);*

*editLoan();*

*}*

*break;*

*case ACTION\_DELETE:*

*deleteSelectedLoans();*

*break;*

*case ACTION\_PRINT:*

*this.model.printLoans();*

*break;*

*case ACTION\_PREVIEW:*

*this.model.previewLoans();*

*break;*

*case ACTION\_ACEPT:*

*if (this.view.isFechaValida()){*

*aceptLoan();*

*UltimaAccion = AccionLoans.NO;*

*this.view.setBotones(0);*

*this.view.limpiaFicha();*

*}*

*break;*

*case ACTION\_CANCEL:*

*UltimaAccion = AccionLoans.NO;*

*this.view.setBotones(0);*

*this.view.limpiaFicha();*

*break;*

*}*

*}*

Este método anterior está sobreescrito puesto que para que la clase controladora pueda escuchar los eventos de la clase vista implementa la interfaz "ActionListener".

También se puede observar que la atención de los eventos de la botonera implica llamadas a métodos de la propia clase y de las clases para vista y modelo. Se explicarán en el apartado "Lógica añadida".

Sustitución del lanzamiento de ventana externa

Para la parte de Préstamos esta ventana no estaba disponible por lo que no tuvimos que eliminarla.

Generación de la lógica de control de la botonera

Por lógica de control de la botonera entendemos la oportuna habilitación y deshabilitación de los botones. Contamos con 7 botones de los cuales 3 (Eliminar, PreView y Print) trabajan por separado y su función no entra en conflicto con el resto. En cambio los botones Añadir, Editar, Aceptar y Cancelar trabajan de manera conjunta y necesitan coordinación.

Dentro de la clase vista LoansManagementView hemos escrito métodos para habilitar y deshabilitar de manera individual y conjunta a estos 4 botones. La lógica del comportamiento conjunto de estos botones está en el método setBotones():

*public void setEnableAdd(){*

*this.btnAdd.setEnabled(true);*

*}*

*public void setEnableEdit(){*

*this.btnEdit.setEnabled(true);*

*}*

*public void setEnableAcept(){*

*this.btnAcept.setEnabled(true);*

*}*

*public void setEnableCancel(){*

*this.btnCancel.setEnabled(true);*

*}*

*public void setDisableAdd(){*

*this.btnAdd.setEnabled(false);*

*}*

*public void setDisableEdit(){*

*this.btnEdit.setEnabled(false);*

*}*

*public void setDisableCancel(){*

*this.btnCancel.setEnabled(false);*

*}*

*public void setDisableAcept(){*

*this.btnAcept.setEnabled(false);*

*}*

*public void setBotones(int inicio){*

*if (inicio == 0){*

*setEnableEdit();*

*setEnableAdd();*

*setDisableCancel();*

*setDisableAcept();*

*} else {*

*setDisableEdit();*

*setDisableAdd();*

*setEnableCancel();*

*setEnableAcept();*

*}*

*}*

Corrección de errores detectados en los métodos

Después de tener una primera implementación del código, pasamos a corregir errores detectados.

Los errores que se detectaron:

* Ordenación alfabética de los combos.
* Fechas de la ficha en blanco.
* Fila tabla Préstamos no seleccionada.

Para la ordenación bastó con incorporar el método estático Collections.sort(). Ejemplo:

*public ArrayList<String> LecturaCustomers() throws CustomersDAOException {*

*ArrayList<String> nombreCustomers = new ArrayList();*

*List<CustomerDTO> listadoCustomers;*

*listadoCustomers=this.custdao.findAll();*

*for (CustomerDTO listCust : listadoCustomers) {*

*nombreCustomers.add(listCust.getApellidos()+" "+listCust.getNombre()+" "+listCust.getIdSocio());*

*}*

*Collections.sort(nombreCustomers);*

*return nombreCustomers;*

*}*

Para las fechas en blanco y fila no seleecionada se incorporaron métodos que indicaban tal estado:

*public boolean isFechaValida() {*

*if (this.getFechaInicio()==-1){*

*JOptionPane.showMessageDialog(this, "Introduzca una fecha de inicio válida", "Ventana de notificación", JOptionPane.ERROR\_MESSAGE);*

*return false;*

*}*

*if (this.getFechaFin()==-1){*

*JOptionPane.showMessageDialog(this, "Introduzca una fecha de final válida", "Ventana de notificación", JOptionPane.ERROR\_MESSAGE);*

*return false;*

*}*

*return true;*

*}*

*public boolean isSelectedRow(){*

*if (this.tblLoans.getSelectedRows().length == 0){*

*JOptionPane.showMessageDialog(this, "Seleccione algún préstamo de la tabla", "Ventana de notificación", JOptionPane.ERROR\_MESSAGE);*

*return false;*

*}*

*return true;*

*}*

Generación de Pruebas Unitarias

Para las pruebas unitarias nos basamos en replicar las clases generadas para las otras funcionalides de Socios y Libros. Se generó una clase para probar métodos de la clase modelo "LoansDAOImpl" y otra clase para probar la vista "LoansTableModel".

Las pruebas se soportan sobre las librerías JUnit y Mockito. Las pruebas sobre la clase vista "LoansTableModel" se realizan con Mockito para evitar utilizar el repositorio original del desarrollo que es MySQL. Mediante la librería hsqldb se genera una base de datos en memoria simulando el entorno de Producción.

##### **Modificaciones realizadas sobre código existente**

Lógica modificada

No hay lógica modificada

Lógica añadida

Hemos añadido código a las clases vista y control, "LoansManagementView" y " LoansManagementController". En la clase de la vista, en la que se utilizan componentes de la clase swing, no se explica la generación automática de código realizada desde el IDE NetBeans.

Clase LoansManagementView

1. Métodos generadores de eventos y su escuchador para los botones Aceptar y Cancelar.

Se suman a los ya existentes para el resto de botones.

*public void setAceptListener(ActionListener l, String command) {*

*this.btnAcept.addActionListener(l);*

*this.btnAcept.setActionCommand(command);*

*}*

*public void setCancelListener(ActionListener l, String command) {*

*this.btnCancel.addActionListener(l);*

*this.btnCancel.setActionCommand(command);*

*}*

2. Métodos generación lógica botonera.

Habilitan y deshabilitan los botnoes de manera individual y conjunta.

*public void setEnableAdd(){*

*this.btnAdd.setEnabled(true);*

*}*

*public void setEnableEdit(){*

*this.btnEdit.setEnabled(true);*

*}*

*public void setEnableAcept(){*

*this.btnAcept.setEnabled(true);*

*}*

*public void setEnableCancel(){*

*this.btnCancel.setEnabled(true);*

*}*

*public void setDisableAdd(){*

*this.btnAdd.setEnabled(false);*

*}*

*public void setDisableEdit(){*

*this.btnEdit.setEnabled(false);*

*}*

*public void setDisableCancel(){*

*this.btnCancel.setEnabled(false);*

*}*

*public void setDisableAcept(){*

*this.btnAcept.setEnabled(false);*

*}*

*public void setBotones(int inicio){*

*if (inicio == 0){*

*setEnableEdit();*

*setEnableAdd();*

*setDisableCancel();*

*setDisableAcept();*

*} else {*

*setDisableEdit();*

*setDisableAdd();*

*setEnableCancel();*

*setEnableAcept();*

*}*

*}*

3. Métodos llamados al pulsar Añadir.

Solo se rellenan los combos con la información de base de datos de socios y libros.

*public void setComboSocios(ArrayList<String> stringo){*

*int numText = stringo.size();*

*jComboBoxSocios.removeAllItems();*

*for (int i=0; i < numText;i++){*

*jComboBoxSocios.addItem(stringo.set(i, stringo.get(i)));*

*}*

*}*

*public void setComboLibros(ArrayList<String> stringo){*

*int numText = stringo.size();*

*jComboBoxLibros.removeAllItems();*

*for (int i=0; i < numText;i++){*

*jComboBoxLibros.addItem(stringo.set(i, stringo.get(i)));*

*}*

*}*

4. Métodos llamados al pulsar Editar.

Se llaman 6 métodos nuevos. Los 2 anteriormente expuestos para rellenar los combos. 2 más para que dichos combos queden seleccionados con el socio y libro del préstamo que se quiere editar. Y por último 2 métodos que recogen las fechas del préstamo y las llevan a la ficha.

*public void selectComboSocios(int id){*

*String strs;*

*StringTokenizer tokens;*

*for (int j=0;j<jComboBoxSocios.getItemCount();j++) {*

*strs = jComboBoxSocios.getItemAt(j).toString();*

*tokens = new StringTokenizer(strs," ");*

*while (tokens.hasMoreTokens()){*

*String ident = tokens.nextToken();*

*if (!(tokens.hasMoreTokens())){*

*if (Integer.parseInt(ident.trim())== id){*

*Object objeto = jComboBoxSocios.getItemAt(j);*

*jComboBoxSocios.setSelectedItem(objeto);*

*jComboBoxSocios.repaint();*

*}*

*}*

*}*

*}*

*}*

*public void selectComboLibros(int id){*

*String strs;*

*StringTokenizer tokens;*

*for (int j=0;j<jComboBoxLibros.getItemCount();j++) {*

*strs = jComboBoxLibros.getItemAt(j).toString();*

*tokens = new StringTokenizer(strs," ");*

*while (tokens.hasMoreTokens()){*

*String ident = tokens.nextToken();*

*if (!(tokens.hasMoreTokens())){*

*if (Integer.parseInt(ident.trim())== id){*

*Object objeto = jComboBoxLibros.getItemAt(j);*

*jComboBoxLibros.setSelectedItem(objeto);*

*jComboBoxLibros.repaint();*

*}*

*}*

*}*

*}*

*}*

*public void setEditFicha(){*

*java.sql.Date fecha = (java.sql.Date)this.tblLoans.getValueAt(this.tblLoans.getSelectedRow(),4);*

*long longfecha = fecha.getTime();*

*setFechaInicio(new Date(longfecha));*

*fecha = (java.sql.Date)this.tblLoans.getValueAt(this.tblLoans.getSelectedRow(),5);*

*longfecha = fecha.getTime();*

*setFechaFin(new Date(longfecha));*

*}*

5. Métodos indicadores de errores.

Métodos que devuelven un valor booleano y en caso de error generan un mensaje indicando el motivo.

*public boolean isFechaValida() {*

*if (this.getFechaInicio()==-1){*

*JOptionPane.showMessageDialog(this, "Introduzca una fecha de inicio válida", "Ventana de notificación", JOptionPane.ERROR\_MESSAGE);*

*return false;*

*}*

*if (this.getFechaFin()==-1){*

*JOptionPane.showMessageDialog(this, "Introduzca una fecha de final válida", "Ventana de notificación", JOptionPane.ERROR\_MESSAGE);*

*return false;*

*}*

*return true;*

*}*

*public boolean isSelectedRow(){*

*if (this.tblLoans.getSelectedRows().length == 0){*

*JOptionPane.showMessageDialog(this, "Seleccione algún préstamo de la tabla", "Ventana de notificación", JOptionPane.ERROR\_MESSAGE);*

*return false;*

*}*

*return true;*

*}*

6. Método utilizado para identificar el préstamo al Editar y Aceptar.

La información del Id del Préstamo no es necesaria al editar el préstamo de manera inicial. Pero cuando ya se acepta la información se debe de identificar el Préstamo y el único modo de hacerlo es capturando este valor sobre la tabla.

*public int getIDPrestamo(){*

*int id = (Integer)(this.tblLoans.getValueAt(getSelectedRow(),1));*

*return id;*

*}*

7. Método para tratamiento de texto mediante clase Locale.

Este método ya existía y se le añadió las líneas que afectaban a los botones Aceptar y Cancelar.

*public void changeLanguage() {*

*LocaleManager lm = LocaleManager.getInstance();*

*this.setTitle(lm.getText("TITLE\_LOANS\_MANAGEMENT"));*

*this.jLabel10.setText(lm.getText("TITLE\_LOANS\_MANAGEMENT"));*

*this.btnAdd.setText(lm.getText("BUTTON\_ADD"));*

*this.btnEdit.setText(lm.getText("BUTTON\_EDIT"));*

*this.btnDelete.setText(lm.getText("BUTTON\_DELETE"));*

*this.btnPreview.setText(lm.getText("BUTTON\_PREVIEW"));*

*this.btnPrint.setText(lm.getText("BUTTON\_PRINT"));*

*this.btnAcept.setText(lm.getText("BUTTON\_OK"));*

*this.btnCancel.setText(lm.getText("BUTTON\_CANCEL"));*

*}*

Clase controladora LoansManagementController

1. Nuevos atributos.

Atributos para mantener la conexión a los modelos Customer y Book.

*private final ICustomersDAO custdao;*

*private final IBooksDAO bookdao;*

Atributos para identificar los eventos del pulsado de los botones Aceptar y Cancelar, y atributos para diferenciar si la acción a ser aceptada es una edición o para añadir un préstamo.

*private static final String ACTION\_ACEPT = "Acept";*

*private static final String ACTION\_CANCEL = "Cancel";*

*private enum AccionLoans { ADD,DELETE,EDIT,PRINT,PREVIEW,NO };*

*private AccionLoans UltimaAccion = AccionLoans.NO;*

2. Métodos de obtención de string compuestos.

Estos métodos utilizan métodos de las clases de los modelos Customer y Book para generar textos compuestos necesarios para rellenar los combos de la ficha. Estos métodos bien podrían estar en las clases modelo.

*public ArrayList<String> LecturaCustomers() throws CustomersDAOException {*

*ArrayList<String> nombreCustomers = new ArrayList();*

*List<CustomerDTO> listadoCustomers;*

*listadoCustomers=this.custdao.findAll();*

*for (CustomerDTO listCust : listadoCustomers) {*

*nombreCustomers.add(listCust.getApellidos()+" "+listCust.getNombre()+" "+listCust.getIdSocio());*

*}*

*Collections.sort(nombreCustomers);*

*return nombreCustomers;*

*}*

*public ArrayList<String> LecturaBooks() throws BooksDAOException {*

*ArrayList<String> tituloBooks = new ArrayList();*

*List<BookDTO> listadoBooks;*

*listadoBooks=this.bookdao.findAll();*

*for (BookDTO listB : listadoBooks) {*

*tituloBooks.add(listB.getNombre()+" "+listB.getIdLibro());*

*}*

*Collections.sort(tituloBooks);*

*return tituloBooks;*

*}*

3. Instanciación de la conexión a los modelos.

Este método constructor de la clase ya estaba generado. Se le ha incluido la instanciación hacia los modelos Customer y Book.

*public LoansManagementController(Connection connection) throws LoansDAOException {*

*if(ConnectionsManager.dbms.equals("oracle")){*

*this.dao = new LoansDAOImplOracle(connection);*

*}*

*else{*

*this.dao = new LoansDAOImpl(connection);*

*}*

*this.model = new LoansTableModel(this.dao, this.dao.findAll());*

*this.view = new LoansManagementView();*

*initView();*

*addListeners();*

*this.view.setVisible(true);*

*LocaleManager.getInstance().addObserver(this);*

*this.bookdao = new BooksDAOImpl (connection);*

*this.custdao = new CustomersDAOImpl (connection);*

*}*

4. Tratamiento de los eventos.

El método ActionPerformed sobreescrito de la implementación de ActionListener ha sido totalmente reescrito para que realice las llamadas a todo el nuevo código generado.

*@Override*

*public void actionPerformed(ActionEvent e) {*

*switch (e.getActionCommand()) {*

*case ACTION\_ADD:*

*UltimaAccion = AccionLoans.ADD;*

*this.view.setBotones(1);*

*rellenaCombos();*

*break;*

*case ACTION\_EDIT:*

*if (this.view.isSelectedRow()){*

*UltimaAccion = AccionLoans.EDIT;*

*this.view.setBotones(1);*

*editLoan();*

*}*

*break;*

*case ACTION\_DELETE:*

*deleteSelectedLoans();*

*break;*

*case ACTION\_PRINT:*

*this.model.printLoans();*

*break;*

*case ACTION\_PREVIEW:*

*this.model.previewLoans();*

*break;*

*case ACTION\_ACEPT:*

*if (this.view.isFechaValida()){*

*aceptLoan();*

*UltimaAccion = AccionLoans.NO;*

*this.view.setBotones(0);*

*this.view.limpiaFicha();*

*}*

*break;*

*case ACTION\_CANCEL:*

*UltimaAccion = AccionLoans.NO;*

*this.view.setBotones(0);*

*this.view.limpiaFicha();*

*break;*

*}*

*}*

5. Método de inicialización.

Ampliamos el método de inicialización con la habilitación inicial para los botones de la ventana de préstamos.

*private void initView() {*

*this.view.changeLanguage();*

*JTable tblLoans = this.view.getTable();*

*tblLoans.setModel(this.model);*

*tblLoans.setRowSorter(new TableRowSorter(this.model));*

*tblLoans.getColumnModel().getColumn(4).setCellEditor(new JDateChooserCellEditor());*

*tblLoans.getColumnModel().getColumn(5).setCellEditor(new JDateChooserCellEditor());*

*((JComponent) tblLoans.getDefaultRenderer(Boolean.class)).setOpaque(true);*

*this.view.setBotones(0);*

*}*

6. Método para que la clase controladora se sume a atender eventos de la vista.

Ampliamos el método siguiente que permite a la clase controladora escuchar los eventos generados al pulsar los botones. Faltaban Aceptar y Cancelar.

*private void addListeners() {*

*this.view.setAddListener(this, ACTION\_ADD);*

*this.view.setEditListener(this, ACTION\_EDIT);*

*this.view.setDeleteListener(this, ACTION\_DELETE);*

*this.view.setPrintListener(this, ACTION\_PRINT);*

*this.view.setPreviewListener(this, ACTION\_PREVIEW);*

*this.view.setAceptListener(this, ACTION\_ACEPT);*

*this.view.setCancelListener(this, ACTION\_CANCEL);*

*}*

7. Método para rellenar los combos con lista cerrada de Socios y Libros.

Este nuevo método obtiene los array de string necesarios para rellenar los combos de la vista.

*private void rellenaCombos() {*

*try{*

*this.view.setComboSocios(LecturaCustomers());*

*} catch (CustomersDAOException ex) {*

*logger.error(ex.toString());*

*}*

*try{*

*this.view.setComboLibros(LecturaBooks());*

*} catch (BooksDAOException ex) {*

*logger.error(ex.toString());*

*}*

*}*

8. Método para leer el contenido de la ficha.

El método getFichaLoan() construye una estructura LoanDTO necesaria tanto para editar o añadir un préstamo, y lo hace con la información de la ficha y de los modelos Customer y Book.

*private LoanDTO getFichaLoan() {*

*LoanDTO getloan = new LoanDTO();*

*CustomerDTO soc = new CustomerDTO();*

*BookDTO book = new BookDTO();*

*try{*

*soc=this.custdao.findByPrimaryKey(this.view.getIDSocio());*

*getloan.setCustomer(soc);*

*} catch (CustomersDAOException ex) {*

*logger.error(ex.toString());*

*}*

*try{*

*book=this.bookdao.findByPrimaryKey(this.view.getIDLibro());*

*getloan.setBook(book);*

*} catch (BooksDAOException ex) {*

*logger.error(ex.toString());*

*}*

*getloan.setFechaInicio(new java.sql.Date(this.view.getFechaInicio()));*

*getloan.setFechaFin(new java.sql.Date(this.view.getFechaFin()));*

*return getloan;*

*}*

9. Método para rellenar la ficha al pulsar el botón Editar.

Método que debe recoger las listas cerradas de los modelos Customer y Book, seleccionar el socio y libro del préstamo y seleccionar las fechas del préstamo.

*private void editLoan() {*

*rellenaCombos();*

*this.view.selectComboSocios(this.model.getLoanAt(this.view.getSelectedRow()).getCustomer().getIdSocio());*

*this.view.selectComboLibros(this.model.getLoanAt(this.view.getSelectedRow()).getBook().getIdLibro());*

*this.view.setEditFicha();*

*}*

10. Método para elegir la opción Editar o Añadir cuando se pulsa el botón Aceptar.

*private void aceptLoan() {*

*switch (UltimaAccion) {*

*case ADD: addFichaLoan();*

*break;*

*case EDIT: editFichaLoan();*

*break;*

*}*

*}*

11. Método que ejecutan Editar o Añadir.

Los 2 métodos siguientes obtienen información de la ficha y ejecutan la acción de Editar o Añadir al modelo.

*private void addFichaLoan(){*

*LoanDTO loan = getFichaLoan();*

*this.model.addLoan(loan);*

*}*

*private void editFichaLoan(){*

*LoanDTO loan = getFichaLoan();*

*loan.setIdPrestamo(this.view.getIDPrestamo());*

*this.model.setLoanAt(loan, this.view.getSelectedRow());*

*}*

##### **Dificultades encontradas**

Entre las dificultades que hemos tenido están:

* Imposibilidad de uso del repositorio SVN: Desde un principio no se pudo usar el repositorio SVN de la UNED para gestionar adecuadamente el desarrollo particular de cada grupo del proyecto. Tuvimos que utilizar el servicio de correo para que Scrum Master juntara todas las piezas.
* Marcar la separación entre modelo, vista y control: A veces la separación del patrón MVC puede tener varias interpretaciones y hay que optar por una de ellas. Cada parte del patrón tiene su responsabilidad y se debe de procurar que la parte controladora esté lo más clara y limpia posible.
* Gestión del desarrollo usando varios IDEs: Las fechas en las que nos encontramos (Semana Santa), han hecho que tengamos migrar el desarrollo a otra plataforma para seguir trabajando amoldándonos a condiciones más duras de trabajo.

## PRESTAMOS, [gbl-2] menu contextual crud

##### **Definición**

Desde ventana principal de administración de Prestamos generar un menú desplegable (combo) con las opciones (Añadir, Editar y Eliminar) al pulsar botón derecho del ratón sobre alguno de los campos del listado de prestamos.

##### **Estrategia de solución**

Basándonos en el desarrollo previo de esta funcionalidad sobre clientes (Costumers) e indicando el código modificado por nuestro Scrum Máster implementar estas modificaciones adecuándolas a las clases y métodos definidas en Loans.

##### **Modificaciones realizadas sobre el código existente**

Lógica Modificada

Clase LoansManagementController.java

Se elimina: protected static final Config config = Config.getInstance();

Por: protected static final LocaleManager lm = LocaleManager.getInstance();

lm es un Atributo de clase. Es una constante de la clase LoansManagementController cuyo atributo es común a todos los objetos de esta clase. Al ser final no admite cambios tras su declaración. Se tiene acceso desde otras subclases.

Se almacena el evento o modificación de estado del objeto que dispara el popup (MVC) al realizar addListeners()

Se elimina

*public LoansManagementController(Connection connection) throws LoansDAOException {*

*//this.model = new LoansModel(connection);*

*if(dbms.equals("oracle")){*

*this.dao = new LoansDAOImplOracle(connection);*

*}*

Por:

*public LoansManagementController(Connection connection) throws LoansDAOException {*

*//this.model = new LoansModel(connection);*

*if(ConnectionsManager.dbms.equals("oracle")){*

*this.dao = new LoansDAOImplOracle(connection);*

*}*

Previamente se importó la librería:

*import es.csc.biblioteca.jdbc.ConnectionsManager;*

Modificación del código para conexión a BBDD Oracle a traves de API jdbc.Se incluye en el método actionPerformed (al que se le pasa como parametro de entrada el evento) la inclusion de ventanas de error cuando al editar un préstamo no se seleccione previamente un registro.

*public void actionPerformed(ActionEvent e) {*

*switch (e.getActionCommand()) {*

*case ACTION\_ADD:*

*addLoan();*

*break;*

*case ACTION\_EDIT:*

*try{*

*editLoan();*

*}catch (IndexOutOfBoundsException ex) {*

*logger.error("Debe seleccionar un registro");*

*JOptionPane.showMessageDialog(this.view,*

*lm.getText("SELECTED\_ROW\_ERROR\_TEXT"),*

*lm.getText("SELECTED\_ROW\_ERROR\_TITLE"),*

*JOptionPane.WARNING\_MESSAGE);*

*}*

*break;*

*case ACTION\_DELETE:*

*deleteSelectedLoans();*

*break;*

*case ACTION\_PRINT:*

*this.model.printLoans();*

*break;*

*case ACTION\_PREVIEW:*

*this.model.previewLoans();*

*break;*

*}*

*}*

Se añaden los Listeners en el método addListeners para el menu popup

*private void addListeners() {*

*this.view.setAddListener(this, ACTION\_ADD);*

*this.view.setEditListener(this, ACTION\_EDIT);*

*this.view.setDeleteListener(this, ACTION\_DELETE);*

*// Añadimos listener para menu contextual*

*this.view.setPopupAddListener(this, ACTION\_ADD);*

*this.view.setPopupEditListener(this, ACTION\_EDIT);*

*this.view.setPopupDeleteListener(this, ACTION\_DELETE);*

*this.view.setPrintListener(this, ACTION\_PRINT);*

*this.view.setPreviewListener(this, ACTION\_PREVIEW);*

*}*

Clase LoansManagementView.java

Se importa la librería java.awt.event.MouseEvent; para gestión del evento del mouse al hacer click en botón derecho.

Incluir en el método initComponents() los nuevos objetos del menu contextual mediante la librería swing. También se incluyen los iconos de anadir, editar y eliminar de Loans.

*private void initComponents() {*

*java.awt.GridBagConstraints gridBagConstraints;*

*jPopupMenuLoans = new javax.swing.JPopupMenu();*

*jMenuItemAddLoan = new javax.swing.JMenuItem();*

*jMenuItemEditLoan = new javax.swing.JMenuItem();*

*jMenuItemDeleteLoan = new javax.swing.JMenuItem();*

*- - - - - - - - - - - - - -*

*- - - - - - - - - - - - - -*

*- - - - - - - - - - - - - -*

*jMenuItemAddLoan.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/book\_blue\_add.png"))); // NOI18N*

*jMenuItemAddLoan.setText("jMenuItem1");*

*jPopupMenuLoans.add(jMenuItemAddLoan);*

*jMenuItemEditLoan.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/book\_blue\_view.png"))); // NOI18N*

*jMenuItemEditLoan.setText("jMenuItem1");*

*jPopupMenuLoans.add(jMenuItemEditLoan); jMenuItemDeleteLoan.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/book\_blue\_delete.png"))); // NOI18N*

*jMenuItemDeleteLoan.setText("jMenuItem2");*

*jPopupMenuLoans.add(jMenuItemDeleteLoan);*

*setClosable(true);*

*setIconifiable(true);*

*setMaximizable(true);*

*setResizable(true);*

*setTitle("Loans Management");*

*getContentPane().setLayout(new java.awt.GridBagLayout());*

*jLabel10.setBackground(new java.awt.Color(64, 64, 64));*

*jLabel10.setFont(new java.awt.Font("Dialog", 1, 18)); // NOI18N*

*jLabel10.setForeground(new java.awt.Color(255, 255, 255));*

*jLabel10.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/books/gui/resources/books32.png"))); // NOI18N*

*jLabel10.setText("Loans Management");*

*jLabel10.setFocusable(false);*

*jLabel10.setName("etiquetaPrestamos"); // NOI18N*

*jLabel10.setOpaque(true);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridwidth = 7;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.ipady = 8;*

*getContentPane().add(jLabel10, gridBagConstraints);*

*jToolBar1.setFloatable(false);*

*jToolBar1.setRollover(true);*

*jToolBar1.setName("botonera"); // NOI18N*

*btnAdd.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/books/gui/resources/book\_blue\_add.png"))); // NOI18N*

*btnAdd.setText("Add");*

*btnAdd.setFocusable(false);*

*btnAdd.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);*

*btnAdd.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);*

*jToolBar1.add(btnAdd);*

*btnEdit.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/book\_blue\_view.png"))); // NOI18N*

*btnEdit.setText("Edit");*

*btnEdit.setFocusable(false);*

*btnEdit.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);*

*btnEdit.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);*

*jToolBar1.add(btnEdit);*

*btnDelete.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/book\_blue\_delete.png"))); // NOI18N*

*btnDelete.setText("Delete");*

*btnDelete.setFocusable(false);*

*btnDelete.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);*

*btnDelete.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);*

*jToolBar1.add(btnDelete);*

*btnAcept.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/printer.png"))); // NOI18N*

*btnAcept.setText("Acept");*

*btnAcept.setFocusable(false);*

*btnAcept.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);*

*btnAcept.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);*

*jToolBar1.add(btnAcept);*

*btnCancel.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/printer.png"))); // NOI18N*

*btnCancel.setText("Cancel");*

*btnCancel.setFocusable(false);*

*btnCancel.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);*

*btnCancel.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);*

*jToolBar1.add(btnCancel);*

*jToolBar1.add(jSeparator1);*

*btnPreview.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/printer\_view.png"))); // NOI18N*

*btnPreview.setText("Preview");*

*btnPreview.setFocusable(false);*

*btnPreview.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);*

*btnPreview.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);*

*jToolBar1.add(btnPreview);*

*btnPrint.setIcon(new javax.swing.ImageIcon(getClass().getResource("/es/csc/biblioteca/loans/gui/resources/printer.png"))); // NOI18N*

*btnPrint.setText("Print");*

*btnPrint.setFocusable(false);*

*btnPrint.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);*

*btnPrint.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);*

*add(btnPrint);*

*gridBagConstraints = new java.awt.GridBagConstraints();*

*gridBagConstraints.gridx = 0;*

*gridBagConstraints.gridy = 1;*

*gridBagConstraints.gridwidth = 7;*

*gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;*

*gridBagConstraints.weightx = 0.1;*

*getContentPane().add(jToolBar1, gridBagConstraints);*

*jScrollPane1.setName("jScrollPane"); // NOI18N*

*tblLoans.setModel(new javax.swing.table.DefaultTableModel(new Object [][] {},*

*new String [] {*

*"Title 1", "Title 2"*

*}*

*));*

Dentro de la initComponents() se hace la llamada a los siguientes tres métodos que controlan los eventos sobre el mouse. Click, Pressed y Released.

*tblLoans.addMouseListener(new java.awt.event.MouseAdapter() {*

*public void mouseClicked(java.awt.event.MouseEvent evt) {*

*tblLoansMouseClicked(evt);*

*}*

*public void mousePressed(java.awt.event.MouseEvent evt) {*

*tblLoansMousePressed(evt);*

*}*

*public void mouseReleased(java.awt.event.MouseEvent evt) {*

*tblLoansMouseReleased(evt)*

Metodos. Tienen como parámetro de entrada un objeto que representa el tipo de evento que se ha producido con el mouse.

*private void tblLoansMouseClicked(java.awt.event.MouseEvent evt) { }*

*Ante este evento no hay código implementado. No debe activarme el menú contextual al realizar click normal (no con botón derecho) con el mouse.*

*private void tblLoansMousePressed(java.awt.event.MouseEvent evt) {*

*if (evt.isPopupTrigger()) {*

*showPopupMenu(evt);*

*} // TODO add your handling code here:*

*}*

*private void tblLoansMouseReleased(java.awt.event.MouseEvent evt) {*

*if (evt.isPopupTrigger()) {*

*showPopupMenu(evt);*

*} // TODO add your handling code here:*

*}*

isPopupTrigger()--> Este método devolverá true si el evento de ratón es el que tiene el sistema operativo por defecto para mostrar los menús emergentes. De esta forma, nuestra aplicación mostrará el menú con el mismo evento de ratón que el resto de las aplicaciones del sistema operativo. El evento es de "mouseReleased" se produce con el botón derecho.

Se añaden las siguientes declaración de variables:

*private javax.swing.JMenuItem jMenuItemAddLoan;*

*private javax.swing.JMenuItem jMenuItemDeleteLoan;*

*private javax.swing.JMenuItem jMenuItemEditLoan;*

*private javax.swing.JPopupMenu jPopupMenuLoans;*

Se incluyen los metodos de control de Listeners para Añadir, Editar y Eliminar

*public void setPopupAddListener(ActionListener l, String command) {*

*this.jMenuItemAddLoan.addActionListener(l);*

*this.jMenuItemAddLoan.setActionCommand(command);*

*}*

*public void setPopupEditListener(ActionListener l, String command) {*

*this.jMenuItemEditLoan.addActionListener(l);*

*this.jMenuItemEditLoan.setActionCommand(command);*

*}*

*public void setPopupDeleteListener(ActionListener l, String command) {*

*this.jMenuItemDeleteLoan.addActionListener(l);*

*this.jMenuItemDeleteLoan.setActionCommand(command);*

*}*

Metodo para mostrar el popup configurando el tamaño y la posición.

*private void showPopupMenu(MouseEvent e) {*

*jPopupMenuLoans.show(this, e.getX()+5, e.getY()+100);*

*}*

Por último, incluir en el método de cambio de idioma las tres opciones posibles en Inglés

*public void changeLanguage() {*

*LocaleManager lm = LocaleManager.getInstance();*

*this.setTitle(lm.getText("TITLE\_LOANS\_MANAGEMENT"));*

*this.jLabel10.setText(lm.getText("TITLE\_LOANS\_MANAGEMENT"));*

*this.btnAdd.setText(lm.getText("BUTTON\_ADD"));*

*this.btnEdit.setText(lm.getText("BUTTON\_EDIT"));*

*this.btnDelete.setText(lm.getText("BUTTON\_DELETE"));*

*this.btnPreview.setText(lm.getText("BUTTON\_PREVIEW"));*

*this.btnPrint.setText(lm.getText("BUTTON\_PRINT"));*

*this.btnAcept.setText(lm.getText("BUTTON\_OK"));*

*this.btnCancel.setText(lm.getText("BUTTON\_CANCEL"));*

*this.jMenuItemAddLoan.setText(lm.getText("MENU\_POPUP\_ADD"));*

*this.jMenuItemEditLoan.setText(lm.getText("MENU\_POPUP\_EDIT"));*

*this.jMenuItemDeleteLoan.setText(lm.getText("MENU\_POPUP\_DELETE"));*

*}*

##### **Dificultades encontradas**

La dificultad surgió a la hora de editar el codigo “protegido” de la clase LoansManagementView.java y más concretamente en el método private void initComponents(). No se encontró la fórmula que dispone Netbeans para poder modificar ese código que genera automáticamente el IDE al diseñar el formulario.

Para poder editarlo se utilizó otro entorno de progración; en nuestro caso DreamWeaver. Se modificó y tras volver abrir el fichero desde Netbeans, se confirma que este código vuelve a ser “protegido” y no permite su edición desde este IDE.

## LIBROS, requisitos [LBR-1] [LBR-2] y [gbl-2]

##### **Especificación de requisito**

Se pide adecuar el proyecto de Biblioteca-Mockito al MVC. Además se pide en la parte de libros asemejar toda la funcionalidad en una sola ficha, respetando el MVC y asemejándola a la práctica de referencia "biblioteca\_ADR".

##### **Estrategia de solución usada**

Para cumplir la especificación, previamente se ha procedido a entender la funcionalidad existente en la práctica de Biblioteca, también hemos adquirido el conocimiento de MVC para adecuar la práctica a la funcionalidad requerida.

Una vez adquirido los conocimientos hemos eliminado del paquete "es.csc.biblioteca.books.gui" la vista y el controlador de la parte Detalle asumiendo dicha funcionalidad la pantalla principal. (Clases BooksManagement).

##### **Modificaciones realizadas sobre código existente**

Lógica Modificada

Eliminación de Clases

BookDetailsController.java

Necesitamos incluir la funcionalidad en una sola ficha/pantalla, de ahí que se eliminase dicha clase insertando los métodos que contenía en la clase BooksManagementController.

BookDetailsView.java

Por el mismo motivo, eliminamos la clase, adquiriendo su funcionalidad (métodos) y la parte diseño la clase "BooksManagementView".

Modificación de Clases

BooksManagementController

Adquiere la funcionalidad/métodos de la clase eliminada BookDetailsController.java.

* + actionPerformed (Gestión de Eventos) Modificamos añadiendo los eventos de los botones Aceptar y Cancelar.
  + getBook
  + checkMandatoryFields
  + doClose

BooksManagementView

Adquiere la funcionalidad/métodos de la clase eliminada BookDetailsView.java

* + setOkListener
  + setCancelListener
  + setBook
  + getBook
  + hideWarnings
  + showWarningTitle
  + showWarningTopic
  + setEstado

##### **Dificultades encontradas**

Nos hemos encontrado problemas con el funcionamiento de los Set Layout.

También nos hemos encontrado problemas al cargar los datos del detalle cuando teníamos

selecionado un libro en el multiregistro.

## socios, [SCS-1] [GBL-3]

##### **Especificación de requisito**

Incluir la modificación de "Socios" en una ficha.

Para ello se ha embebido la ventana en la interfaz donde se muestra la tabla.

Referencias para impelementar el código fuente:

<http://62.204.199.127/JAVA_UNED/bea/browser/biblioteca_Mockito>

##### **Estrategia de solución usada**

Para implementar el código, se han modificado las siguientes clases:

1. Para la parte de funcionalidad:
   1. CustomerManagmentController.java
   2. CustomerManagmentView.java
   3. CustomerDetailsController.java (se deja de usar)
   4. CustomerDetailsView.java (se deja de usar)
2. Para la parte de pruebas unitarias:
   1. CustomerDAOTest.java
   2. CustomerTableModelTest.java

##### **Lógica Modificada**

CustomerManagmentController.java

Se han añadido a la importación los siguientes paquetes nuevos:

*import es.csc.biblioteca.customers.dao.CustomersDAOImplOracle;*

*import static es.csc.biblioteca.customers.gui.CustomerDetailsController.RET\_OK;*

*import static es.csc.biblioteca.customers.gui.CustomerDetailsController.lm;*

*import es.csc.biblioteca.exceptions.MandatoryFieldException;*

*import es.csc.biblioteca.gui.main.TableRow;*

*import es.csc.biblioteca.util.Config;*

*import es.csc.biblioteca.jdbc.ConnectionsManager;*

Creamos la clase CustomersManagementController:

*public class CustomersManagementController implements ActionListener, Observer {*

*protected static final Logger logger = Logger.getLogger(CustomersManagementController.class);*

*private final ICustomersDAO dao;*

*private final CustomersManagementView view;*

*private final CustomersTableModel model;*

*private static final String ACTION\_ADD = "Add";*

*private static final String ACTION\_EDIT = "Edit";*

*private static final String ACTION\_DELETE = "Delete";*

*private static final String ACTION\_PRINT = "Print";*

*private static final String ACTION\_PREVIEW = "Preview";*

*private static final String ACTION\_OK = "Ok";*

*private static final String ACTION\_CANCEL = "Cancel";*

*public static final int RET\_CANCEL = 0;*

*public static final int RET\_OK = 1;*

*private int selectedRow;*

*public CustomersManagementController(Connection connection) throws CustomersDAOException {*

*if(ConnectionsManager.dbms.equals("oracle")){*

*this.dao = new CustomersDAOImplOracle(connection);*

*}*

*else{*

*this.dao = new CustomersDAOImpl(connection);*

*}*

*this.dao = new CustomersDAOImpl(connection);*

*this.model = new CustomersTableModel(this.dao, this.dao.findAll());*

*this.view = new CustomersManagementView();*

*this.view.changeLanguage();*

*this.view.setTableModel(this.model); //.getTableModel());*

*this.view.setRowSorter(new TableRowSorter(this.model)); //.getTableModel()));*

*this.view.setVisible(true);*

*addListeners();*

*LocaleManager.getInstance().addObserver(this);*

*}*

*@Override*

*public void actionPerformed(ActionEvent e) {*

*switch (e.getActionCommand()) {*

*case ACTION\_ADD:*

*addCustomer();*

*capturaCampo();*

*break;*

*case ACTION\_EDIT:*

*editCustomer();*

*break;*

*case ACTION\_DELETE:*

*deleteSelectedCustomers();*

*break;*

*case ACTION\_PRINT:*

*this.model.printCustomers();*

*break;*

*case ACTION\_PREVIEW:*

*this.model.previewCustomers();*

*break;*

Aquí se añaden los dos controles. Para el "OK" y el "CANCEL":

*case ACTION\_OK:*

*try {*

*CustomerDTO dto = this.view.getCustomer();*

*checkMandatoryFields(dto);*

*this.modelDTO = dto;*

*this.model.setCustomerAt(this.getCustomer(),this.selectedRow);*

*this.view.borrar();*

*} catch (MandatoryFieldException ex) {*

*logger.error("Faltan datos obligatorios.");*

*JOptionPane.showMessageDialog(this.view,*

*lm.getText("MANDATORY\_FIELDS\_ERROR\_TEXT"),*

*lm.getText("MANDATORY\_FIELDS\_ERROR\_TITLE"),*

*JOptionPane.WARNING\_MESSAGE);*

*}*

*break;*

*case ACTION\_CANCEL:*

*this.view.borrar();*

*break;*

*}*

*}*

*private void addListeners() {*

*this.view.setAddListener(this, ACTION\_ADD);*

*this.view.setEditListener(this, ACTION\_EDIT);*

*this.view.setDeleteListener(this, ACTION\_DELETE);*

Aquí se añaden los listeners para el menú contextual:

*this.view.setPopupAddListener(this, ACTION\_ADD);*

*this.view.setPopupEditListener(this, ACTION\_EDIT);*

*this.view.setPopupDeleteListener(this, ACTION\_DELETE);*

*this.view.setPrintListener(this, ACTION\_PRINT);*

*this.view.setPreviewListener(this, ACTION\_PREVIEW);*

Después se añaden los listeners para la nueva zona:

*this.view.setOkListener(this, ACTION\_OK);*

*this.view.setCancelListener(this, ACTION\_CANCEL);*

*}*

*public CustomersManagementView getView() {*

*return this.view;*

*}*

*private void addCustomer() {*

Abrimos la ventana para dar de alta a un usuario:

*CustomerDetailsController controller = new CustomerDetailsController();*

*if (controller.getReturnStatus() == CustomerDetailsController.RET\_OK) {*

*this.model.addCustomer(controller.getCustomer());*

*int index = this.model.getRowCount();*

*this.view.setSelectedRow(index - 1);*

*}*

*}*

*private void editCustomer() {*

*if(this.view.getSelectedRow()!=-1)*

*{*

*this.selectedRow = this.view.getSelectedRow();*

*CustomerDTO dto = this.model.getCustomerAt(selectedRow);*

*this.view.setCustomer(dto);*

*}*

*else*

*{*

*logger.error("No has seleccionado ningún usuario");*

*JOptionPane.showMessageDialog(this.view,*

*lm.getText("NO\_SELECTION\_ERROR\_TITLE"),*

*lm.getText("NO\_SELECTION\_ERROR\_TEXT"),*

*JOptionPane.WARNING\_MESSAGE);*

*}*

*}*

*private void deleteSelectedCustomers() {*

*JOptionPane p = new JOptionPane();*

*this.model.deleteSelectedCustomers();*

*}*

*@Override*

*public void update(Observable o, Object arg) {*

*if (o instanceof LocaleManager) {*

*this.view.changeLanguage();*

*}*

*}*

Añadimos las funciones:

*public CustomerDTO getCustomer() {*

*return modelDTO;*

*}*

Se comprueba el valor de los campos obligatorios:

*private void checkMandatoryFields(CustomerDTO customer) throws MandatoryFieldException {*

*Boolean showWarnings = Boolean.FALSE;*

*this.view.hideWarnings();*

*if (customer.getApellidos() == null || "".equals(customer.getApellidos().trim())) {*

*showWarnings = Boolean.TRUE;*

*this.view.showWarningLastName(Boolean.TRUE);*

*}*

*if (customer.getDireccion() == null || "".equals(customer.getDireccion().trim())) {*

*showWarnings = Boolean.TRUE;*

*this.view.showWarningAddress(Boolean.TRUE);*

*}*

*if (customer.getDni() == null || "".equals(customer.getDni().trim())) {*

*showWarnings = Boolean.TRUE;*

*this.view.showWarningNIC(Boolean.TRUE);*

*}*

*if (customer.getNombre() == null || "".equals(customer.getNombre().trim())) {*

*showWarnings = Boolean.TRUE;*

*this.view.showWarningFirstName(Boolean.TRUE);*

*}*

*if (customer.getFechaAlta() == null) {*

*showWarnings = Boolean.TRUE;*

*this.view.showWarningDate(Boolean.TRUE);*

*}*

*if (showWarnings) {*

*throw new MandatoryFieldException();*

*}*

*}*

*}*

CustomerManagmentView.java

Se han añadido a la importación los siguientes paquetes nuevos

*import es.csc.biblioteca.customers.dao.CustomerDTO;*

*import java.awt.event.MouseEvent;*

*import java.sql.Date;*

Se ha añadido el método hideWarnings(); dentro de la clase CustomersManagementView:

*public class CustomersManagementView extends JInternalFrame {*

*public CustomersManagementView() {*

*initComponents();*

*hideWarnings();*

*}*

Se ha modificado el código original del método llamado desde dentro del constructor para iniciar el formulario:

*@SuppressWarnings("unchecked")*

*private void initComponents() {*

*java.awt.GridBagConstraints gridBagConstraints;*

*jSplitPane1 = new javax.swing.JSplitPane();*

*jPopupMenuCustomers = new javax.swing.JPopupMenu();*

*jMenuItemAddCustomer = new javax.swing.JMenuItem();*

*jMenuItemEditCustomer = new javax.swing.JMenuItem();*

*jMenuItemDeleteCustomer = new javax.swing.JMenuItem();*

*jLabel10 = new javax.swing.JLabel();*

*jToolBar1 = new javax.swing.JToolBar();*

*btnAdd = new javax.swing.JButton();*

*btnEdit = new javax.swing.JButton();*

*btnDelete = new javax.swing.JButton();*

*jSeparator1 = new javax.swing.JToolBar.Separator();*

*btnPreview = new javax.swing.JButton();*

*btnPrint = new javax.swing.JButton();*

*jScrollPane1 = new javax.swing.JScrollPane();*

*tblCustomers = new javax.swing.JTable();*

Se han mantenido los botones de "Add", "Edit" "Delete", "Preview" y "Print".

(...)

Aquí se añaden los dos nuevos botones:

*public void setOkListener(ActionListener l, String command) {*

*this.btnOk.addActionListener(l);*

*this.btnOk.setActionCommand(command);*

*}*

*public void setCancelListener(ActionListener l, String command) {*

*this.btnCancel.addActionListener(l);*

*this.btnCancel.setActionCommand(command);*

*}*

*public void setTableModel(AbstractTableModel m) {*

*this.tblCustomers.setModel(m);*

*}*

*public void setRowSorter(TableRowSorter sorter) {*

*this.tblCustomers.setRowSorter(sorter);*

*}*

*public int getSelectedRow() {*

*if(this.tblCustomers.getSelectedRow()==-1)*

*{*

*return -1;*

*}*

*else*

*{*

*return this.tblCustomers.convertRowIndexToModel(this.tblCustomers.getSelectedRow());*

*}*

*}*

*public void setSelectedRow(int indexRow) {*

*int index = this.tblCustomers.convertRowIndexToView(indexRow);*

*this.tblCustomers.setRowSelectionInterval(index, index);*

*}*

*private void showPopupMenu(MouseEvent e) {*

*jPopupMenuCustomers.show(this, e.getX()+5, e.getY()+100);*

*}*

*public void changeLanguage() {*

*LocaleManager lm = LocaleManager.getInstance();*

*this.setTitle(lm.getText("TITLE\_CUSTOMERS\_MANAGEMENT"));*

*this.jLabel10.setText(lm.getText("TITLE\_CUSTOMERS\_MANAGEMENT"));*

*this.btnAdd.setText(lm.getText("BUTTON\_ADD"));*

*this.btnEdit.setText(lm.getText("BUTTON\_EDIT"));*

*this.btnDelete.setText(lm.getText("BUTTON\_DELETE"));*

*this.btnPreview.setText(lm.getText("BUTTON\_PREVIEW"));*

*this.btnPrint.setText(lm.getText("BUTTON\_PRINT"));*

*this.jMenuItemAddCustomer.setText(lm.getText("MENU\_POPUP\_ADD"));*

*this.jMenuItemEditCustomer.setText(lm.getText("MENU\_POPUP\_EDIT"));*

*this.jMenuItemDeleteCustomer.setText(lm.getText("MENU\_POPUP\_DELETE"));*

Se añade la nueva parte.

*this.setTitle(lm.getText("TITLE\_CUSTOMER\_DETAILS"));*

*this.jLabel10.setText(lm.getText("TITLE\_CUSTOMER\_DETAILS"));*

*this.lblDNI.setText(lm.getText("LABEL\_DNI"));*

*this.lblFirstName.setText(lm.getText("LABEL\_FIRST\_NAME"));*

*this.lblLastName.setText(lm.getText("LABEL\_LAST\_NAME"));*

*this.lblAddress.setText(lm.getText("LABEL\_ADDRESS"));*

*this.lblCreationDate.setText(lm.getText("LABEL\_CREATION\_DATE"));*

*this.btnOk.setText(lm.getText("BUTTON\_OK"));*

*this.btnCancel.setText(lm.getText("BUTTON\_CANCEL"));*

*}*

Añadimos las funciones que leen y escriben los campos de texto.

*public CustomerDTO getCustomer() {*

*CustomerDTO dto = new CustomerDTO();*

*if (!"".equals(this.lblIdCustomer.getText())) {*

*dto.setIdSocio(Integer.parseInt(this.lblIdCustomer.getText()));*

*}*

*dto.setNombre(this.txtFirstName.getText());*

*dto.setApellidos(this.txtLastName.getText());*

*dto.setDireccion(this.txtAddress.getText());*

*dto.setDni(this.txtDNI.getText());*

*if (this.dcCreationDate.getDate() != null) {*

*dto.setFechaAlta(new Date(this.dcCreationDate.getDate().getTime()));*

*}*

*return dto;*

*}*

*public void hideWarnings() {*

*this.lblWarningAddress.setVisible(false);*

*this.lblWarningDate.setVisible(false);*

*this.lblWarningFirstName.setVisible(false);*

*this.lblWarningLastName.setVisible(false);*

*this.lblWarningNIC.setVisible(false);*

*}*

*public void showWarningAddress(Boolean visible) {*

*this.lblWarningAddress.setVisible(visible);*

*}*

*public void showWarningDate(Boolean visible) {*

*this.lblWarningDate.setVisible(visible);*

*}*

*public void showWarningFirstName(Boolean visible) {*

*this.lblWarningFirstName.setVisible(visible);*

*}*

*public void showWarningLastName(Boolean visible) {*

*this.lblWarningLastName.setVisible(visible);*

*}*

*public void showWarningNIC(Boolean visible) {*

*this.lblWarningNIC.setVisible(visible);*

*}*

*public void setCustomer(CustomerDTO customer) {*

*if (customer == null) {*

*this.dcCreationDate.setDate(new java.util.Date());*

*this.lblIdCustomer.setText(null);*

*this.txtFirstName.setText(null);*

*this.txtLastName.setText(null);*

*this.txtDNI.setText(null);*

*this.txtAddress.setText(null);*

*this.dcCreationDate.setDate(null);*

*} else {*

*this.lblIdCustomer.setText(String.valueOf(customer.getIdSocio()));*

*this.txtFirstName.setText(customer.getNombre());*

*this.txtLastName.setText(customer.getApellidos());*

*this.txtDNI.setText(customer.getDni());*

*this.txtAddress.setText(customer.getDireccion());*

*this.dcCreationDate.setDate(customer.getFechaAlta());*

*}*

*}*

Finalmente, se crea la función encargada de borrar los campos:

*public void borrar() {*

*this.dcCreationDate.setDate(new java.util.Date());*

*this.lblIdCustomer.setText(null);*

*this.txtFirstName.setText(null);*

*this.txtLastName.setText(null);*

*this.txtDNI.setText(null);*

*this.txtAddress.setText(null);*

*this.dcCreationDate.setDate(null);*

*}*

*}*

Se han dejado de usar las clases CustomerDetailsController.java y CustomerDetailsView.java, y para la parte de pruebas unitarias se han modificado las clases CustomerDAOTest.java y CustomerTableModelTest.java. A continuación se explican los cambios en estas dos últimas clases.

DAOTest.java

Se establece una conexión con la base de datos de pruebas:

*private static Connection getConnection() throws SQLException {*

*return DriverManager.getConnection("jdbc:hsqldb:mem:.;shutdown=true", "SA", "");*

*}*

Se crea la base de datos de pruebas:

*private static void createDatabase() throws Exception {*

*Statement statement = connection.createStatement();*

*for (String sql : getSQLStatements("uned\_creation\_hsqldb.sql").split(";;")) {*

*if (sql.length() > 0) {*

*try {*

*statement.execute(sql);*

*} catch (Exception e) {*

*System.err.println("Failed to execute SQL statement " + sql);*

*throw e;*

*}*

*}*

*}*

*}*

Aquí se insertan los datos de las pruebas en la base de datos:

*private static void populateDatabase() throws Exception {*

*Statement statement = connection.createStatement();*

*for (String sql : getSQLStatements("uned\_inserts\_hsqldb.sql").split(";;")) {*

*if (sql.length() > 0) {*

*try {*

*statement.execute(sql);*

*} catch (Exception e) {*

*System.err.println("Failed to execute SQL statement " + sql);*

*throw e;*

*}*

*}*

*}*

*}*

Se obtienen las sentencias SQL del fichero indicado

*private static String getSQLStatements(String file) throws IOException {*

*String retval = "";*

*final byte[] buf = new byte[4096];*

*final InputStream is = CustomersDAOTest.class.getResourceAsStream(file);*

*while (is.read(buf) > 0) {*

*retval += new String(buf);*

*}*

*return retval.trim();*

*}*

Creamos un socio para usar en las pruebas:

*private static CustomerDTO getSocio() {*

*CustomerDTO socio = null;*

*SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");*

*try {*

*socio = new CustomerDTO();*

*socio.setNombre("David");*

*socio.setApellidos("Atencia");*

*socio.setDireccion("Nerja");*

*socio.setDni("53155478V");*

*socio.setFechaAlta(new Date(formatter.parse("2015-11-21").getTime()));*

*} catch (ParseException ex) {*

*}*

*return socio;*

*}*

CustomerTableModelTest.java

Se ha importado el nuevo paquete:

*import java.lang.Boolean;*

Se ha añadido la variable private List<CustomerDTO> al; dentro de la clase CustomersTableModelTest:

*@RunWith(MockitoJUnitRunner.class)*

*public class CustomersTableModelTest {*

*private static ICustomersDAO mockedCustomersDAO;*

*private static CustomersTableModel instance;*

*private List<CustomerDTO> al;*

*public CustomersTableModelTest() {*

*}*

Se ha incluído un System.out.println para mostrar por consola que se empieza el análisis.

*@BeforeClass*

*public static void setUpClass() throws CustomersDAOException {*

*System.out.println("Comenzamos el Análisis");*

*System.out.println();*

*// Create mock object of ICustomersDAO*

*mockedCustomersDAO = mock(ICustomersDAO.class);*

*when(mockedCustomersDAO.findAll()).thenReturn(new ArrayList<CustomerDTO>());*

*when(mockedCustomersDAO.insert((CustomerDTO) anyObject())).thenReturn(1);*

*doNothing().when(mockedCustomersDAO).delete(anyInt());*

*when(mockedCustomersDAO.findByPrimaryKey(anyInt())).thenReturn(null);*

*doNothing().when(mockedCustomersDAO).update(anyInt(), (CustomerDTO) anyObject());*

*}*

Se ha añadido un System.out.println para mostrar por consola cuando acaba la prueba.

*@AfterClass*

*public static void tearDownClass() {*

*System.out.println();*

*System.out.println("Fin de la prueba");*

*}*

Aquí se hace lo mismo para mostrar por consola que se empieza el test.

*@Before*

*public void setUp() {*

*System.out.println("Comenzamos el test");*

*al = new ArrayList<CustomerDTO>();*

*for(int i = 1; i <= 10; i++) {*

*CustomerDTO dto = new CustomerDTO();*

*dto.setIdSocio(i);*

*dto.setNombre("Nombre\_" + i);*

*dto.setApellidos("Apellidos\_" + i);*

*dto.setDireccion("Dirección\_" + i);*

*dto.setDni("DNI\_" + i);*

*dto.setFechaAlta(null);*

*al.add(dto);*

*}*

*// Create objet to test*

*instance = new CustomersTableModel(mockedCustomersDAO, al);*

*}*

Aquí finalmente mostramos por consola que es el fin del test.

*@After*

*public void tearDown() {*

*System.out.println("Fin del test");*

*}*

*/\*\**

*\* Test of getRowCount method, of class CustomersTableModel.*

*\* @throws es.csc.biblioteca.customers.dao.CustomersDAOException*

*\*/*

*@Test*

*public void testGetRowCount() throws CustomersDAOException {*

*System.out.println("getRowCount");*

*int expResult = 10;*

*int result = instance.getRowCount();*

*assertEquals(expResult, result);*

*}*

*/\*\**

*\* Test of getColumnCount method, of class CustomersTableModel.*

*\*/*

*@Test*

*public void testGetColumnCount() {*

*System.out.println("getColumnCount");*

*int expResult = 7;*

*int result = instance.getColumnCount();*

*assertEquals(expResult, result);*

*}*

Posteriormente se han descomentado los métodos siguientes para que funcione correctamente toda la clase.

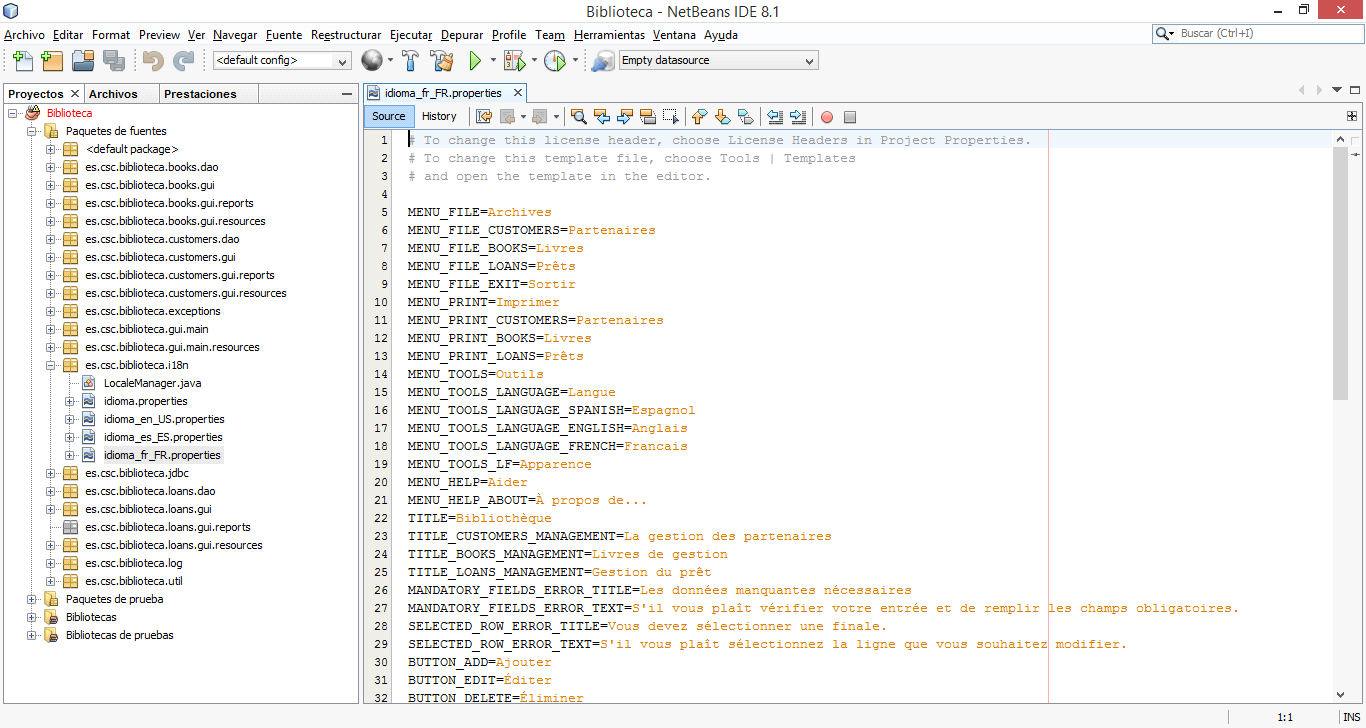
Internacionalización

Se ha exigido como requisito de la aplicación, la inclusión del idioma francés a la aplicación. Evidentemente este paso se ha al finalizar el resto de requisitos, ya que si no, toda cosa que se cree posteriormente a esta internacionalización no quedará recogida en ella.

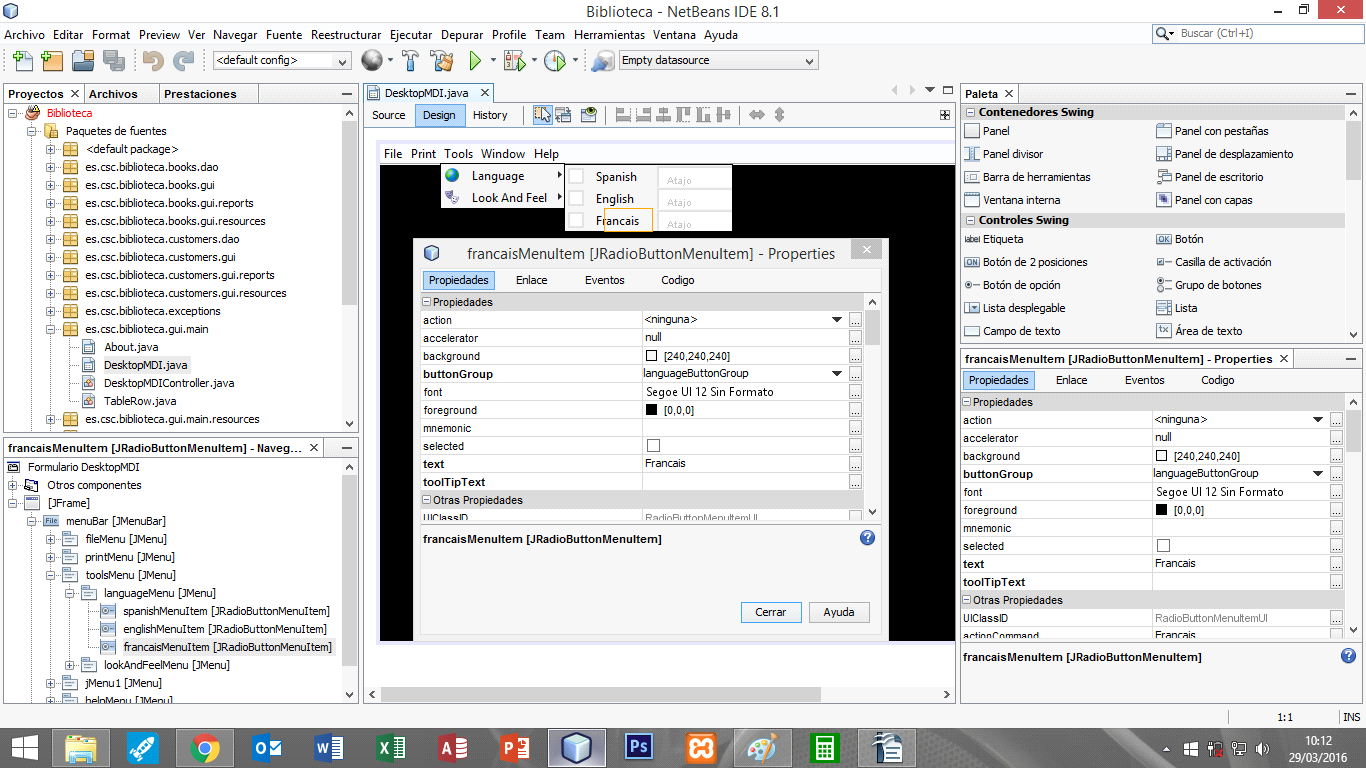
Los pasos necesarios para llevar a cabo esta tarea son:

Se crea un nuevo archivo .properites llamado idioma\_fr\_FR.properties dentro del paquete es.csc.biblioteca.i18n. La manera más sencilla de comenzar es copiar el contenido de otro archivo de idioma dentro de este para tener ya todas las etiquetas que deberán ser traducidas. Una vez hecho esto, con paciencia, hay que traducir dichas etiquetas al idioma destino, en este caso el francés.

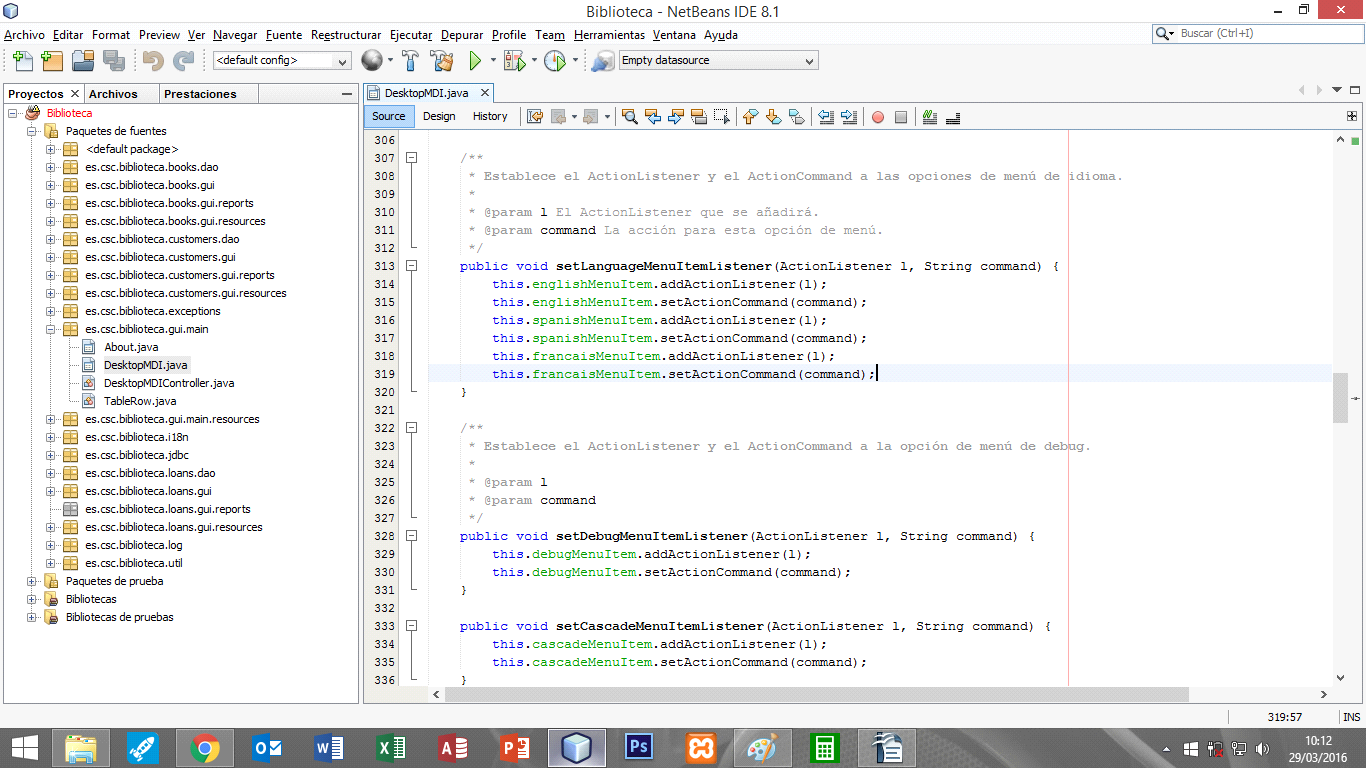
A parte de traducir las etiquetas que ya existieran es necesario añadir en este archivo y en el resto de idiomas una nueva línea en la que se va a generar el texto de la nueva etiqueta, a la hora de elegir el idioma en el menú. En la imagen posterior se encuentra en la línea 18.



Una vez hecho esto es necesario acudir al editor swing que trae netbeans y añadir al grupo de idiomas la opción de elegir el idioma en francés. La manera más sencilla es copiar una de las dos opciones que ya existen y modificar las propiedades para lograr nuestro objetivo.



Ahora tenemos que dotar de vida este nuevo radioButton para que responda cuando sea seleccionado, para ello en primer lugar vamos a añadir los escuchadores de eventos necesarios, para que reaccione a ser seleccionado.



Finalmente debemos desarrollar el código con las acciones necesarias como respuesta a estos eventos.

